Source Code Overlay Editor

ABSTRACT

The concept of operations for a new type of editor is described. The source code overlay editor (SCOE) allows a user to improve computer source code artifacts by providing links, replacement text, and the means to show or hide design and implementation details by traversing links and causing text substitutions based on replacement text. Among other things these capabilities give the author/reader an ability to use hyperlinked replacement text substitutions to hide or reveal either source code or abstractions of source code as is most appropriate for the reader's current context (level of understanding and purpose).

These methods will provide the programmer a means to better indicate why source code statements are present and how a desired effect is implemented. The SCOE method offers a new type of visualization ability to the programmer and source code maintainer that will facilitate problem decomposition and documentation.

1.0 INTRODUCTION

"Software visualization is the systematic and imaginative use of the technology of interactive computer graphics and the disciplines of graphic design, typography, color, cinematography, animation, and sound design to enhance the comprehension of algorithms and computer programs" (1). The following paragraphs detail a proposed methodology that is a new method for software visualization.

The four elements that are the core of the proposed SCOE technology, macro expansion, function abstraction, hypertext linkage, and Graphical User Interface's, are not new to computer science. Today however they have not been unified into the SCOE method described here, e.g., they have not been used in the following manner to better our abilities to create and maintain computer programs.

The programmer can use the function abstraction provided through the programming language and macro expansion as provided by preprocessor tool to create arbitrarily high or low (de)composition of a set of actions as described in source code narrative. However both suffer serious problems when used in a 'flat file sense' because navigating through source code text is awkward and the reader is easily confused due to lexical scoping and cognitive effects. Further, parameter declarations are error prone and time-consuming, making function abstraction an awkward means for such composition. The proposed SCOE method leverages a common technology to offer another way to represent, visualize and navigate source code artifacts. This new method allows the reader to decide the level of detail that is appropriate for the current level of investigation.

2.0 REFERENCES

1. Software Visualization for Debugging, R. Baecker, C. DiGiano, A. Marcus, Communications of the ACM, April 1997, Volume 40, Number 4, page 44.

3.0 DESCRIPTION OF SCOE (METHOD)

Figure 1 illustrates the process of defining replacement text, the basic functional element of SCOE operation. Figures 2 and 3 illustrate some potential applications of the method. By allowing the programmer to determine the contents of each replacement this editor provides the programmer with the ability to

- associate human-oriented descriptive text with source statements
- replace source statements or replacement text with replacement text

T. Risko 2-29-01 4/9

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE Professional Paper	3. DATES COVERED

4. TITLE AND SUBTITLE Source Code Overlay Editor	5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Theodore F. Risko	5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Air Warfare Center Aircraft Division 22347 Cedar Point Road, Unit #6 Patuxent River, Maryland 20670-1161	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Naval Air Systems Command 47123 Buse Road Unit IPT Patuxent River, Maryland 20670-1547	10. SPONSOR/MONITOR'S ACRONYM(S) 11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The concept of operations for a new type of editor is described. The source code overlay editor (SCOE) allows a user to improve computer source artifacts by providing links, replacement text, and the means to show or hide design and implementation details by traversing links and causing text substitutions based on replacement. Among other things these capabilities give the author/reader an ability to use hyperlinked replacement text substitutions to hide or reveal either source code or abstractions of source code as is most appropriate for the reader's current context (level of understanding and purpose).

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Theodore F. Risko
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (include area code)
Unclassified	Unclassified	Unclassified	Unclassified	6	(301) 757- 3475 / 342-2368

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

- eliminate (hide) source statements or replacement text

Through such capabilities the SCOE allows a user to tailor the contents of the artifact so that the reader is presented with those details as is most appropriate for the level of understanding or state of the examination. As a result, while reading an artifact the user may select an abstraction of interest (text for which replacement text exists) and progressively examine finer levels of details leading ultimately to source code. Similarly, the user could navigate the links in the opposite direction and move toward higher and more human-oriented descriptions.

Navigating these links provides the user with the means for focusing on detail as is most appropriate. As an example, a programmer could define a highest level of logical detail by using the appropriate replacements to eliminate variable declarations and/or initializations or the details of error exception handling. Such streamlining offers the reader a clear picture of the algorithmic details within a program.

4.0 CONCLUSIONS

When replacement text is used to describe program logic and actions at varying levels of detail then traversing this series of text equivalents allows a reader to view the "program source code" at the level of detail most appropriate for presenting the relevant aspects of a problem at a current level of understanding. That is, the editor allows examination of detail that is at the context of the current state of knowledge or area of focus of the reader. Through such editing and text representation techniques the SCOE provides the programmer with new and improved abilities to create artifacts that may more clearly communicate functions, structure, and purpose such as:

- create source code files that allow one to determine the purpose of source code statements, by examining the associated human-oriented replacements that correspond to selected areas

- determine how desired outcomes stated at a high level of abstraction have been implemented using program language statements

- selectively and optionally provide or delete text not directly related to understanding, for example variable declarations within a function abstraction

- providing a means to allow descriptions of programs at a high level provides a means for rapid familiarization with a programs organization and features. As Figure 3 illustrates such high level descriptions allow a new reader to rapidly determine the area responsible for a particular action, and the implementation details that determine how that section of code operates. Following linked replacement text in turn allows determining the appropriate details at any level as is selected during the examination.

- Besides such singular, individual substitutions it is also possible to provide higher level, derived capabilities such as groupings that associate several replacements all of a given type (for example all of a given level of replacement). Such sets of multiple replacements could be used to create a view - by conglomerating numerous replacements one could create a block replacement that causes the entire artifact to be described at a specified level of detail. By giving the reader the ability to traverse these various views of program source code the SCOE system may be used to facilitate composition, decomposition and documentation - it allows creation of views of the program that are most appropriate for the level of understanding and stage of examination as is occurring.

(Re)describing program text in these ways offers the ability to create improved documentation systems thereby facilitating program maintenance. Having these editing and association capabilities available within a code development editor/workstation should lead to numerous related capabilities as illustrated by the following examples:

T. Risko 7-29-[illegible] 5/9

- providing a means for focusing attention on module interfaces and by so doing facilitate open-system approaches to software development

- supporting methods for facilitating reverse engineering of computer programs

- devising methods for documenting higher level abstractions such as architectures and data and control flow

- creating text systems that better provide and interrelate programs to their documentation

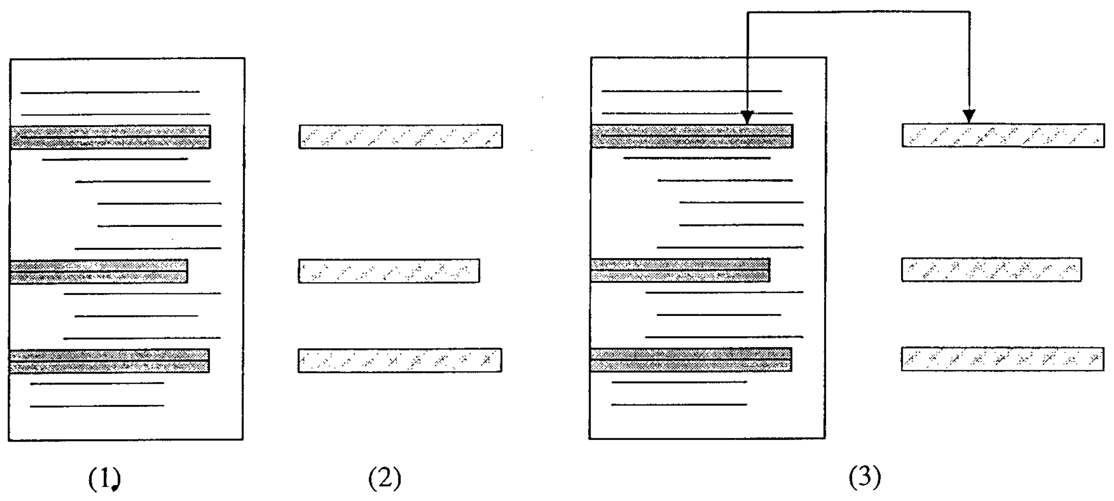
- exploration and research, such as on developing a generic programming language; this may ultimately allow programming (more) independently from a computer source code languages

- through tagging of source code statements the identification of a certain class of statements. This may facilitate the identification, location and inspection, removal or insertion of sets of statements added for a purpose or with a set of changes. Obviously such tags would be invisible to the reader unless the reader requested them to be visible e.g., during editing or inspection of tags.

- using these concepts within non-programming documents for tracking the source of or details on the elements of a document

Because hyperlinking allows parameterless text replacement the method will be easy to use. Since the content of the replacement text is not necessarily bound by syntax rules this method gives the user the ability to easily associate user-defined and human-oriented descriptions to the source statement(s). This new type of visualization created by the SCOE can be used to improve both the clarity of program source code artifacts and the means by which these artifacts may be created and documented.

T. Risko 2-29-00 6/9



This figure shows how a view of an area of code may be created by identifying the text to be associated with a replacement, entering replacement text, and associating the replacement text with the area of interest. Note that a single selection of text is comprised of non-contiguous text.

(1) Select text that is to have an associated replacement.

(2) Enter a replacement text.

(3) Indicate the mapping of the replacement to the original.

Figure 1 - Basic SCOE Operation

T.Risko 2-29-00 7/9

Example 1.

```
Source code selected for replacement:
if (!p_tbl->CMyDBTbl1:: GetTblData ()) {
  err_string = "Failure Reading Tbl1";
  return (FALSE); }
if (!p_tbl->CMyDBTbl1:: GetTblData ()) {
  err_string = "Failure Reading Tbl2";
  return(FALSE); }
```

```
Using the SCOE associate the following:

  Read_ SubTables _Tbl1_and_Tbl2
```

Example 2.

```
Source code selected for replacement:
   angle = angle / 360.0;
   iangle  = (int)angle + 15;
   iangle  = iangle  * 30;
   angle = (double)  iangle ;
```

```
Using the SCOE associate the following:

  Truncate_and_Increment_Angle;
```

Examples of how the SCOE may streamline code through basic substitution. Using such associations allows creating human-oriented source artifacts that state (parts of) the program at a higher level of abstraction than program source code.

Figure 2 - Code Streamlining

T. Risko 2/29/00 8/9

Level 1.

```
command_processor:
  while (the current command is not the EXIT command) {
    prompt for a command
    read users command
    parse the command to determine if it   containd an opt par
    call the command handler
  }
}
```

Level 2.

```
command_processor:

// while the current command is not the exit command
while(cmd_id != EXIT_CMD) {
  // prompt for a command
    prepare to prompt
    display prompt

  //read users command
    fgets(input_data,STRING_SIZE-2, stdin)
    determine if end of input or input error occurred

  //parse the command to determine if it contained an opt par
    if (input_was_received) {
      prepare_to_parse
      parse_the_command

  //call the command handler
    switch (cmd_id) {
    case GET_SYS_ID:
      call get_sys_id_ cmd function
    case HELP_CMD:
      call help_ cmd function
    default:
      indicate invalid command was input
    }  // switch
  } // if
} // while
}
```

Level 3.

```
void command_processor() {
CMD_TYPE cmd_id = NO_CMD;
char input_data[STRING_SIZE];

// while the current command is not the exit command
while(cmd_id != EXIT_CMD) {
  // prompt for a command
    // prepare to prompt
      strcpy(cmd_parameter,"");
      sprintf(svar,"%s CMD>", PROGRAM_NAME);

    // display prompt
      w_out(svar,STRING_SIZE,WRITE_STDOUT,"CPROC");_
  // read users command
    if (fgets(input_data,STRING_SIZE-2, stdin) == NULL) {

    //determine if end of input or input error occurred
      switch( ferror(stdin)) {
      case 0:
        sprintf(svar,"%s END OF INPUT",PROGRAM_NAME);
        cleanup(CLEAN_EXIT, svar);
      default:
        sprintf(svar,"%s INPUT ERROR",PROGRAM_NAME);
        cleanup(ERROR_EXIT, svar);
      }
    }
  //parse the command to determine if it contained an opt par
    if (strlen(input_data) > 0) < // input was received
      prepare_to_parse
      parse_the_command

  // call the command handler
    switch (cmd_id) {
    case GET_SYS_ID:
      call get_sys_id_ cmd_function
    case HELP_CMD:
      call help_ cmd function
    default:
      indicate invalid command was input
    }  // switch
  } // if
} // while
}
```

Three views of a function illustrate how problem decomposition may be facilitated though user-defined text replacements. In this example the Level 3 view is part source code and part replacement text, illustrating that an editor may allow replacements of selected text portion(s). Showing the replacement text of other levels as comments is for illustration only.

Figure 3 - Problem Decomposition

T.Risko 2/29/00 9/9